第 33 卷 第 11 期 计 算 机 工 程 2007 年 6 月
Vol.33 No.11 Computer Engineering June 2007
·人工智能及识别技术· 文章编号：1000—3428(2007)11—0193—03 文献标识码：A 中图分类号：TP311

微粒群算法中惯性权重的调整策略

胡建秀，曾建潮
(太原科技大学系统仿真与计算机应用研究所，太原 030024)

摘 要：惯性权重是微粒群算法中的关键参数，可以平衡算法全局搜索能力和局部搜索能力的关系，提高算法的收敛性能。该文分析了惯性权重对微粒群算法收敛性能的影响，为了进一步提高算法的全局最优性，提出了几种对惯性权重的调整策略。通过对 4 个测试函数的仿真实验，验证了这些策略的可行性，表明这些策略能够简便高效地提高算法的全局收敛性和收敛速度。

关键词：微粒群算法；惯性权重；全局最优性

Selection on Inertia Weight of Particle Swarm Optimization

HU Jianxiu, ZENG Jianchao
(Institute of System Simulation & Computer Application, Taiyuan University of Science and Technology, Taiyuan 030024)

【Abstract】The inertia weight is the crucial parameter of the particle swarm optimization(PSO). It can balance the global search and local search to improve PSO's convergence. This paper analyzes the effect of inertia weight on PSO's performance. To enhance the global optimality, a few adjusting methods on inertia weight are put forward. The results on four benchmark functions prove these methods are feasible, and indicate these methods can improve the global convergence and convergence speed.

【Key words】Particle swarm optimization; Inertia weight; Global optimality

1995 年，James Kennedy和Russell Eberhart提出了微粒群算法(PSO算法)，它是一种基于群体的、自适应的搜索优化方法。其基本思想是受他们早期对许多鸟类的群体社会行为进行建模和仿真结果的启发，主要利用的模型是生物学家Frank Heppner的鸟类模型[1,2]。

与其它进化算法相类似，微粒群算法也采用了“群体”和“进化”的概念，也通过个体间的协作和竞争，实现复杂空间中最优解的搜索。但微粒群算法不像其他进化算法那样对个体使用进化算子，而是将每个个体看成是在n维作搜索空间中的一个没有重量和体积的微粒，以一定的速度向适应度最好的微粒位置飞行[9]。

微粒群算法已被作为解决复杂优化问题的一种有效方法。但它也存在精度不高、易发散等缺点，为此，许多改进方法将微粒群算法同遗传思想、模拟退火算法和单纯性方法等结合起来，以期提高算法的全局搜索能力和精度[2~4]。

本文根据惯性权重对微粒群算法优化性能影响的规律[5,8]，提出了微粒群算法中惯性权重的几种选择策略。通过实验测试，表明这些策略能够使算法有更好的全局最优性，同时加快了算法的收敛速度。

1 微粒群算法与惯性权重

1.1 基本微粒群算法

微粒群算法的基本原理：将所优化问题的每一个解称为一个微粒，每个微粒在n维搜索空间中以一定的速度飞行，通过适应度函数来衡量微粒的优劣，微粒根据自己的飞行经验以及其他微粒的飞行经验动态调整飞行速度，以期向群体中最好微粒位置飞行，使所优化问题得到最优解[1,2]。

微粒群算法的具体实现步骤如下：

Step1 初始化一群微粒，包括微粒的随机位置和速度；

Step2 根据适应度函数计算每个微粒的适应值；

Step3 对每个微粒，将其适应值与所经历过的最好位置 pbest 作比较，如果较好，则将其作为当前的最好位置 pbest；

Step4 对每个微粒，将其适应值与全局所经历的最好位置 gbest 作比较，如果较好，则将其作为当前的全局最好位置；

Step5 根据式(1)、式(2)对微粒的速度和位置进行进化；

Step6 如未达到结束条件(通常为足够好的适应值或达到一个预设最大代数 Gmax)，则返回 Step2。

基本微粒群算法的进化方程式(1)、式(2)如下：

$$v_i(t+1)=v_i(t)+c_1r_1(p_i-x_i(t))+c_2r_2(p_g-x_i(t)) \quad (1)$$

$$x_i(t+1)=v_i(t+1)+x_i(t) \quad (2)$$

1.2 微粒群算法中的惯性权重

为了改善基本微粒群算法的收敛性能，Y. Shi和R. C. Eberhart在 1998 年IEEE国际进化计算学术会议上发表了题为“A Modified Particle Swarm Optimizer”的论文，在速度进化方程中引入惯性权重[5]，即

$$v_i(t+1)=wv_i(t)+c_1r_1(p_i-x_i(t))+c_2r_2(p_g-x_i(t)) \quad (3)$$

式中 w 称为惯性权重，它使微粒保持运动惯性，使其有扩展搜索空间的趋势，有能力搜索新的区域。

在式(3)中，第 1 部分用于保证算法的全局收敛性能；第 2、第 3 部分使得算法具有局部搜索能力，引入惯性权重w可以平衡全局搜索能力和局部搜索能力的比例关系，提高算法的性能。w值较大时，全局搜索能力强，局部搜索能力弱；w值较小时，反之。一般地，在全局搜索算法中，希望前期有较高的搜索能力以得到合适的种子，而在后期有较高的开发

基金项目：教育部重点科研基金资助项目(204018)
作者简介：胡建秀(1978－)，女，硕士生，主研方向：系统建模与仿真；曾建潮，教授、博导
收稿日期：2006-06-30 **E-mail：**blueblue_bird@163.com

能力，以加快收敛速度。Y. Shi和R. C. Eberhart建议在微粒群算法中采用线性递减权重策略[5,6]，即

$$w^k = w_{ini} - \frac{(w_{ini} - w_{end})}{K_{\max}} \times k \tag{4}$$

其中 k 为当前进化代数，Kmax 为最大进化代数，wini 为初始惯性权重值，wend 为进化至最大代数时的惯性权重值。目前，较典型的取值方法是 wini=0.9，wend=0.4。这种选取策略对某些优化问题会得到很好的效果，但对于不同的问题，算法中每一代所需要的比例关系并不相同，在此，本文提出了惯性权重的其他的几种策略方法，通过典型函数的测试，表明这些策略能够使得微粒群算法解决一些问题时有更好的全局收敛性。

2 惯性权重的几种调整策略

策略 1 典型的线性递减

$$w^k = w_{ini} - \frac{(w_{ini} - w_{end})}{K_{\max}} \times k \tag{5}$$

策略 2 线性微分递减

$$\frac{dw}{dk} = -\frac{2(w_{ini} - w_{end})}{k_{\max}^2} \times k \tag{6}$$

$$w^k = w_{ini} - \frac{(w_{ini} - w_{end})}{K_{\max}^2} \times k^2 \tag{7}$$

策略 3 非线性微分变化

$$\frac{dw}{dk} = \frac{(w_{ini} - w_{end})}{k_{\max}} - \frac{4(w_{ini} - w_{end})}{k_{\max}^2} \times k \tag{8}$$

$$w^k = w_{ini} + \frac{(w_{ini} - w_{end})}{K_{\max}} \times k - \frac{2(w_{ini} - w_{end})}{K_{\max}^2} \times k^2 \tag{9}$$

策略 4 步长较小的线性递减

$$w^k = w_{ini} - \frac{(w_{ini} - w_{end})}{K_{\max}^2} \times k \tag{10}$$

示意图例如图 1~图 4 所示。

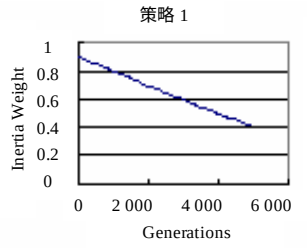

图 1 策略 1

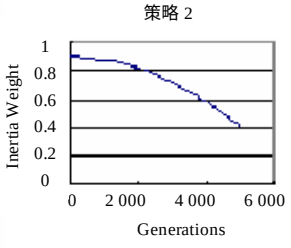

图 2 策略 2

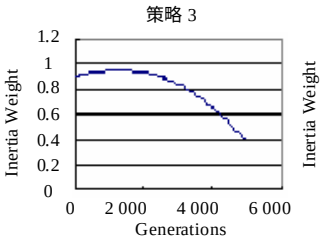

图 3 策略 3

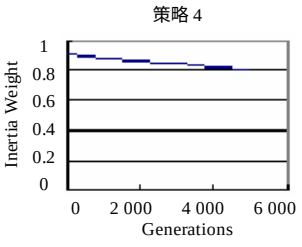

图 4 策略 4

在上述方法中，策略 1(典型线性递减策略)是目前较为通用的取法，但是，在算法进化初期，全局搜索能力较强，如果在初期搜索不到最好点，随着 w 的减小，局部搜索能力加强，易陷入局部极值。在分析这种缺陷的基础上，本文针对性地提出了 3 种策略：

(1)是策略 2(线性微分递减策略)，通过对 w 的变化方程的分析以及示意图所示可以看到，在算法进化初期，w 的减小趋势缓慢，全局搜索能力很强，有利于找到很好的优化种子，在算法进化后期，w 的减小趋势加快，一旦在前期找到合适的种子，可以使得算法收敛速度加快。

(2)策略 3(非线性微分变化策略)，在这种方法中，算法开始时，w 先逐渐增加到某个预设代数(如 Kmax/4)，然后再微分递减。这种策略可以在算法初期加大搜索力度，有利于找到最好点。

(3)策略 4(步长较小的线性递减策略)，步长较小，则 w 的变化幅度较小，不易陷入局部最优。

3 实验仿真

下面通过 4 个典型测试函数的仿真实验，来测试上述选择策略的性能。4 个测试函数分别如下：

f_1：是著名的Sphere函数，单峰，在x_i=0 时达到极小值。

$$f_1(x) = \sum_{i=1}^{n} x_i^2 \qquad -100 \le x_i \le 100 \tag{11}$$

f_2：被称为Rosenbrock函数，非凸、病态、单峰，在x_i=1 时达到极小值。

$$f_2(x) = \sum_{i=1}^{n} (100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2) \qquad -30 \le x_i \le 30 \tag{12}$$

f_3：被称为Rastrigin函数，多峰，有很多正弦凸起的局部极小点，在x_i=0 时达到全局极小点。

$$f_3(x) = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10) \qquad -5.12 \le x_i \le 5.12 \tag{13}$$

f_4：是著名的Schaffer函数，由J. D. Schaffer提出，全局极大点是(0,0)，多峰，在距离全局极大点大约 3.14 的范围内存在无限多的次全局极大点。

$$f_4(x) = 0.5 - \frac{(\sin^2\sqrt{x_1^2 + x_2^2} - 0.5)}{(1 + 0.001(x_1^2 + x_2^2))^2}\sqrt{a^2 + b^2} \qquad -100 \le x_i \le 100 \tag{14}$$

在对这 4 个测试函数优化的标准PSO算法中，取c1=1.8；c2=1.8；f_1，f_4中最大进化代数取为 1 000，群体规模为 30，误差为 0.000 1；f_2，f_3中最大进化代数取为 5 000，群体为 50，误差为 0.1；f_1，f_2，f_3的测试维数为 10，f_4的测试维数为 2；各个函数的vmax取变量范围的上限；算法的实验次数为 50。

在典型线性递减方法中，取 wini=0.9，wend=0.4；在后面 3 种方法中由于算法初期 w 的变化很缓慢，全局搜索能力持续较强，因此 wini 可以相对取小一些，这里选取 2 组值：

(1)wini=0.8，wend =0.5；

(2)wini=0.6，wend=0.4；

分别进行实验，结果如表 1~表 4 所示。

表 1 Sphere 函数

w 的选择策略	收敛率	平均收敛代数	平均最好值
策略 1(0.9-0.4)	50/50	483.0	0.000 086
策略 2(0.8-0.5)	50/50	547.2	0.000 082
策略 3(0.8-0.5)	50/50	782.5	0.000 080
策略 4(0.8-0.5)	5/50	978.6	0.001 597
策略 2(0.6-0.4)	50/50	138.3	0.000 082
策略 3(0.6-0.4)	50/50	146.4	0.000 085
策略 4(0.6-0.4)	50/50	139.1	0.000 084

表 2 Rosenbrock 函数

w 的选择策略	收敛率	平均收敛代数	平均最好值
策略 1(0.9-0.4)	15/50	3 248.8	1.147 046
策略 2(0.8-0.5)	9/50	3 178.5	1.386 455
策略 3(0.8-0.5)	6/50	4 196.0	1.977 152
策略 4(0.8-0.5)	3/50	1 423.3	4.215 714
策略 2(0.6-0.4)	15/50	2 334	1.105 219
策略 3(0.6-0.4)	24/50	2 455	0.943 248
策略 4(0.6-0.4)	35/50	2 892.2	0.726 172

表 3 Rastrigin 函数

w 的选择策略	收敛率	平均收敛代数	平均最好值
策略 1(0.9-0.4)	13/50	2 463.1	1.311 295
策略 2(0.8-0.5)	15/50	2 620.9	0.915 040
策略 3(0.8-0.5)	22/50	3 702.6	0.842 470
策略 4(0.8-0.5)	47/50	3 188.6	0.130 406
策略 2(0.6-0.4)	1/50	396.0	4.618 336
策略 3(0.6-0.4)	2/50	240.0	4.818 533
策略 4(0.6-0.4)	0/50	——	4.437 515

表 4 Schaffer 函数

w 的选择策略	收敛率	平均收敛代数	平均最好值
策略 1(0.9-0.4)	41/50	453.6	0.001 788
策略 2(0.8-0.5)	46/50	395.2	0.000 829
策略 3(0.8-0.5)	47/50	441.3	0.000 635
策略 4(0.8-0.5)	47/50	385.3	0.000 508
策略 2(0.6-0.4)	24/50	206.5	0.005 077
策略 3(0.6-0.4)	30/50	199.8	0.003 915
策略 4(0.6-0.4)	31/50	191.9	0.003 723

从上面的结果中，可以看到：

(1)在标准微粒群算法中，惯性权重取典型的 0.9 递减到 0.4，对某些问题并不能够达到很好的收敛效果，尤其是对于典型的 Rosenbrock 函数和 Rastrigin 函数，算法的收敛率不高，并且所取得平均最好值离最优点偏差很大。

(2)对于上述 4 个函数，分别选用了 2 组参数(0.8-0.5，0.6-0.4)来进行比较测试，对于单峰函数f_1、f_2，惯性权重采用 0.6-0.4 时，3 种惯性权重策略效果较好，对于多峰函数f_3、f_4，采用 0.8-0.5 效果更好。这个结论首先表明惯性权重采用线性微分递减、非线性微分变化以及步长较小的线性递减都能使算法得到优于典型线性递减的收敛性能，收敛率、收敛速度以及所得的平均最好值方面都有一定的改善。另外，也表明了这些选取策略对于多峰函数和单峰函数有些差别，由于在优化过程中，多峰函数很容易陷入局部最优，因此，w取值相对于单峰大些，递减速度缓慢一些，能使算法收敛性能良好。

(3)Rosenbrock 函数和 Rastrigin 函数一直都是很难优化的复杂函数，当采用步长较小的线性递减策略时，这两个函数在合适的参数下都能得到很显著的效果，如表 2 中，w 从 0.6 以步长 $(w_{ini}-w_{end})/K_{\max}^2$ 线性递减到 0.4 时，收敛率为 35/50，相对于从 0.9 以步长 $(w_{ini}-w_{end})/K_{\max}$ 线性递减到 0.4 的收敛率提高了很多。而表 3 中，采用较小步长线性递减时，收敛率为 47/50，全局最优性较好，同时所得到的平均最好值也更接近函数的最好点。所以，对较难优化的问题，可以尝试采用惯性权重 w 减小趋势相对缓弱的方法，使得算法进化过程初、中期保持较强的全局搜索能力。

4 结论

微粒群算法是一种相对新型的、有潜力的演化算法，但目前在算法收敛性和精度方面还存在一些问题。本文在分析微粒群算法原理和惯性权重对其收敛性能的影响情况的基础上，提出了几种惯性权重的改善方法，利用 4 个经典测试函数进行实验，其中尤其 Rosenbrock 函数和 Rastrigin 函数利用通常的线性递减方法，全局收敛性较差。实验结果表明，这些改善方法能够很好改进算法的收敛性能。同时，实验结果表明在对多峰函数和单峰函数的优化过程中，由于函数本身的特点，所适合的参数选择有所差异。

目前，国内外学者对微粒群算法进行了多方面的研究，比如对微粒群算法的收敛性和收敛速度估计方面的尝试性数学证明[2,7]，以及补充和扩展微粒群算法与其它技术的结合来解决算法易陷入局部最优的问题[10~12]等。笔者认为利用不同类型问题的特点来改进微粒群算法或者设计有一定针对性的算法，极具研究意义。

参考文献

1 Kennedy J, Eberhart R C. Particle Swarm Optimization[C]//Proc. of IEEE Int'l. Conf. on Neural Networks, Piscataway. IEEE Service Center, 1995: 1942-1948.

2 曾建潮, 介　婧, 崔志华. 微粒群算法[M]. 北京: 科学出版社, 2004-05.

3 高　尚, 杨静宇, 吴小俊, 等. 基于模拟退化算法思想的粒子群优化算法[J]. 计算机应用与软件, 2005, 22(1): 103-104.

4 杨亚平, 曾建潮. 微粒群与单纯形相结合的混合优化[C]//中国模糊逻辑与计算智能联合学术会议. 2005: 804-807.

5 Shi Y , Eberhart R C. A Modified Particle Swarm Optimization[C]//Proceedings of the Congress on Evolutionary Computation, Piscataway. IEEE Press, 1998: 69-73.

6 Shi Y, Eberhart R C. Empirical Study of Particle Swarm Optimization[C]//Proceedings of the Congress on Evolutionary Computation, Piscataway. IEEE Service Center, 1999: 1945-1950.

7 曾建潮, 崔志华. 一种保证全局收敛的 PSO 算法[J]. 计算机研究与发展, 2004, 41(8): 1333-1338.

8 王俊伟, 汪定伟. 粒子群算法中惯性权重的实验与分析[J]. 系统工程学报, 2005, 20(2): 194-198.

9 谢晓锋, 张文俊, 杨之廉. 微粒群算法综述[J]. 控制与决策, 2003, 18(2): 129-134.

10 Ratnaweera A, Halgamuge S K, Watson H C. Self-organizing Hierarchical Particle Swarm Optimizer With Time-varying Acceleration Coefficients[J]. IEEE Transactions on Evolutionary Computation, 2004, 8(3): 240-255.

11 Xie X F, Zhang W J, Yang Z L. A Dissipative Particle Swarm Optimization[C]//Proceedings of the Congress on Evolutionary Computation (CEC), Hawaii, USA. 2002: 1456-1461.

12 Zhang W J, Xie X F. DEPSO: Hybrid Particle Swarm with Differential Evolution Operator[C]//Proc. of IEEE Int. Conf. on Systems, Man & Cybernetics, Washington D C, USA. 2003: 3816-3821.